IoT Network Monitoring and Security System

Submitted by:
Martin Atanacio
T00684924
Bachelors of Computing Science
Thompson Rivers University
26 Nov. 2023

Supervisor:
Anthony Aighobahi
Computing Science
Thompson Rivers University

805 TRU Way
Kamloops, BC, V2C 0C8

Abstract

This project involves demonstrating an approach to building a network monitoring and security system using the Internet of Things (IoT) environment. Our team created a system consisting of a Raspberry Pi integrated with various IoT devices, such as LEDs, buzzers, temperature and humidity sensors, ultrasonic sensors, and a keypad, that will be used to enhance the system's capabilities. Network activity is captured using Wireshark/Tshark, a commonly used packet analysis tool that provides our team with the data that we need. All data is sent to our real-time Firebase database which is interconnected with our website that helps users visualize this data. Thus, improving the overall monitoring experience and also facilitating real-time decision making. Overall, the project demonstrates the implementation of IoT devices alongside the Wireshark tool as a scalable solution to network monitoring.

Table of Contents

Summary

Problem

This project addresses the issues found in network monitoring and the overall security of a system within a topology. By increasing the systems found within a network, we are also increasing the chances of attacks to that network. Needless to say, it is crucial to protect that network and secure it both internally as well as physically. The management of these systems is important in order to manage the flow of data and the external conditions that may affect the system's usage. Currently, there is a lack of user-friendly interfaces that allow network administrators to visualize the data collected by the network as well as data collected from the system's external environment.

Method of Investigation

The project focused on designing an Internet of Things (IoT) system composed of a Raspberry Pi as the central hub interconnected with various IoT devices - like LEDs, buzzers, temperature and humidity sensors, ultrasonic sensors, and a keypad - to enhance the system's security and the response capabilities. Moreover, we used the Wireshark packet analysis tool to capture network data, which will be sent to our real-time Firebase database for centralized storage and retrieval. Lastly, the creation of a user-friendly website allows for graphical visualizations of this data which can ultimately help network administrators analyze the data at a glance and make better informed decisions.

Conclusions

In conclusion, our project achieved the successful development and implementation of an IoT based network monitoring system. Integrating different IoT devices certainly made our system more flexible in the way we collected data. Wireshark turned out to be a great tool for analyzing network traffic and collecting the results that we needed. Using Firebase not only made things more organized, but it also allowed an easy way for our website to display our results.

Recommendations

Based on our findings, we suggest refining and optimizing the system itself to keep up with security threats, as it is necessary to expand on the continuous updates and improvements to the integration of said IoT devices and prevent them from being considered a liability in the network topology. Additionally, future research could focus on the integration of machine learning to further intensify the system's capabilities to adapt and respond to any network challenges encountered.

Introduction

Subject

The investigation revolves around the development and implementation of a network monitoring and security system that uses IoT devices to enhance the systems capabilities. Our aim is to provide a comprehensive understanding of the general challenges within network monitoring, emphasizing the need for modern solutions that integrate IoT technologies. Optimizing computer resources is crucial in today's world, as larger networks imply a bigger risk factor for network security. Implementing more systems in a network is considered a liability, as it now poses an additional point of vulnerability to the whole system. Being able to analyze the traffic and access to these systems can help any organization manage and control their devices and their usage.

Purpose

The primary purpose of this investigation is to design, implement, and evaluate an IoT based network monitoring system that is able to offer real time insights into the system's network activities as well as external conditions that may be affecting the usage of the systems. By developing a system that monitors network transmissions and actively responds to anomalies/threats, we aim to provide a contribution to the field of network security and inform network administrators about an affordable and effective solution using modern IoT technology.

Scope

The scope of this project is outlined by the design of the proposed IoT system. We mainly focused on integrating various IoT components, such as sensors, that can be used as an additional type of insight that can later be used to analyze the system's external conditions, such as human presence and authorization into the system. However, the system is prone to several limitations, including both hardware and security limitations. The adaptation of IoT technology poses a risk in network security due to the several types of attacks that can be conducted on the different interconnected devices. Our results from this investigation aim to define the boundaries in existing network monitoring systems as well as on our approach, while also providing a baseline for future developments and research in IoT security.

Methods, Assumptions and Procedures

Methods

The problem emerged from the need to measure the network traffic and the conditions of each computer system inside the computer science support lab on our campus. The need for this came from the desire to prove to the university's board that the classroom was indeed being used frequently by students, with the aim of getting extra funds to allow for more systems to be added and thus increasing the number of people coming in for help. In order to achieve this, we decided that the best way to prove our needs was by creating a network monitoring system that could allow us to measure the network traffic and availability in the room. With this in mind, we could use our tool to analyze what people are doing in the support lab, how many users we got coming on a daily basis, and how many computers are available for students to come in and work. Any anomalies or unsuspected activity could also be a raise for concern and inform the network administrators about any issues we encountered.

In terms of tools, the heart of the project corresponds to Wireshark, a network tool utilized for packet capturing and analysis. In order to use it in our Raspberry PI, we used a Python library tool called Pyshark that calls the functionalities of Tshark, a UNIX based tool from Wireshark that will grant us the ability to run Wireshark in the background and save this data into a file. Pyshark will then allow us to convert this data into a JSON (JavaScript Object Notation) file. Data that is in JSON format can easily be sent to and received by our Firebase real time database. This database runs in the cloud and is responsible for organizing the data received. Additionally, Firebase is integrated with our web interface, where data will be extracted and converted into meaningful visual graphs.

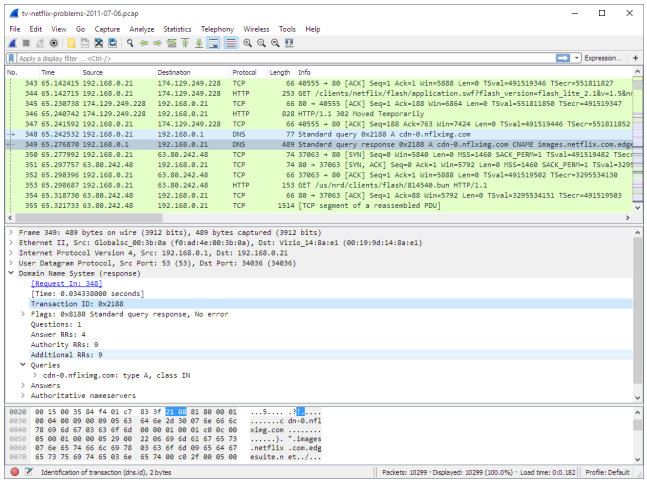

Figure 1.1: Wireshark Tool

Figure 1.2: Firebase Real-Time Database

In terms of hardware equipment, the project consists of a variety of IoT devices connected to our Raspberry PI, each serving an important function to the system. In order to measure the presence of an individual in the system, we used an ultrasonic sensor that is able to determine when a user is in the proximity of a computer in the support lab. A keypad is used to grant access to the room and help prevent unauthorized access. If, however, an unauthorized access has been detected, a buzzer will sound and an LED will turn on to indicate a potential intruder. Lastly, a temperature/humidity sensor is used to detect any drastic changes in the environment. This will help us detect unusual conditions in the room or even detect if a system is overheating.

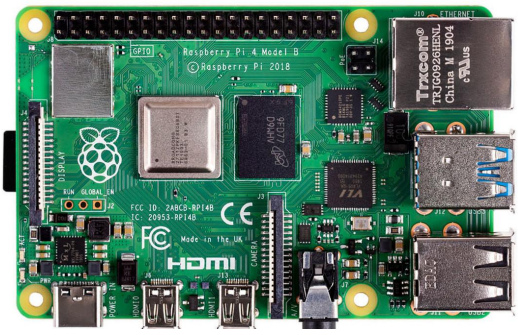

Figure 1.3: Raspberry PI

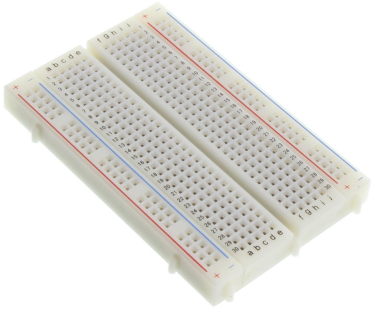

Figure 1.4: Breadboard

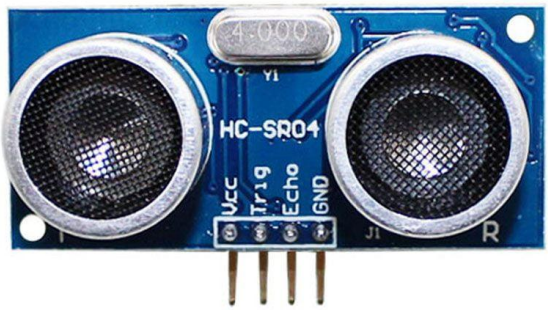

Figure 1.5: HC-SR04 Ultrasonic Distance Sensor

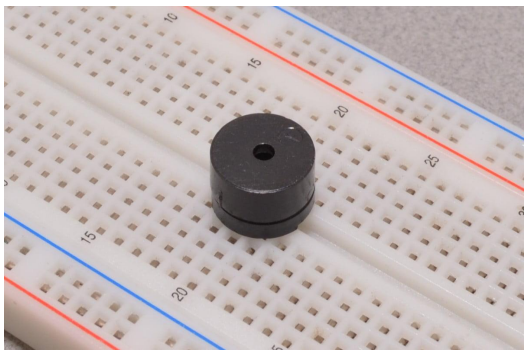

Figure 1.6: Buzzer

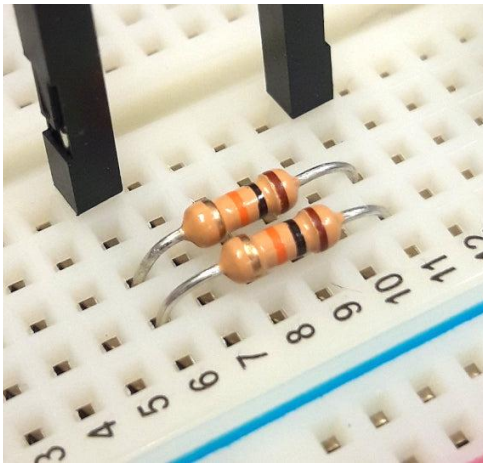

Figure 1.7: Resistor

Figure 1.8: Keypad

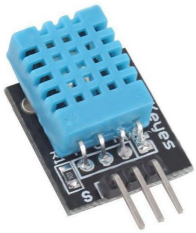

Figure 1.9: DHT11 Temperature/Humidity Sensor

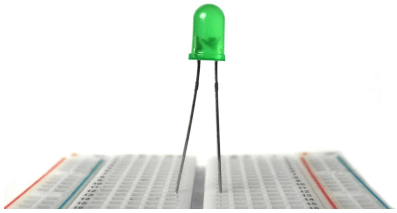

Figure 1.10 LED Light

Assumptions

Some assumptions had to be made about the conditions of the support lab so that we can implement the system effectively. For instance, we are relying on capturing network data at a constant fixed rate, therefore an important assumption to be made is that the lab has access to a stable and functional network that will also support the deployment of IoT devices. Additionally, we are making the assumption that the layout of the support lab will remain the same, thus excluding the chances that the ultrasonic sensor can measure inaccurate and undesired data. Another assumption to be considered is that the IT department allows for the necessary configurations and support to deploy the system within the existing network.

These assumptions serve as planning tools for the implementation of the IoT system inside a lab on campus. It's important to validate these assumptions and if necessary, make adjustments given the specific limitations of the support lab environment.

Procedures

In order to understand the problem and challenges associated with network monitoring and security, we first had to analyze the requirements that we needed that would help us arrive to the conclusions that we wanted. To gain a better understanding of how we could collect data from the network, we delved into the documentation of networking tools, specifically Wireshark. By exploring some of the functionalities and capabilities of Wireshark, we were able to quickly arrange a solution. The Pyshark Python library can be used with Tshark, a UNIX tool that allows us to run Wireshark in the background, automating the process of network data collection. By actively testing and engaging with the abilities of Tshark, we were able to understand how to approach this project.

Through our previous experience, we determined that the data collected by all our devices, including Tshark, should be sent to a cloud based database. This way, all our data is in an organized centralized location, which will then aid us when pulling that data and displaying it in our website. In order to receive the data in Firebase, we first had to convert it to a JSON format, given that Tshark would provide us with a different format. Using Python programs, the same process would apply to all our sensors and devices. They will connect to our Firebase database and push the data that they measured. Similarly, devices such as the LED and the buzzer will act accordingly depending on the data read from the database. Throughout this process, a lot of testing was involved and refining of the system before we were certain that we were producing the correct results. Once the results were validated, we were now able to connect our database with our web interface and display all of our data:

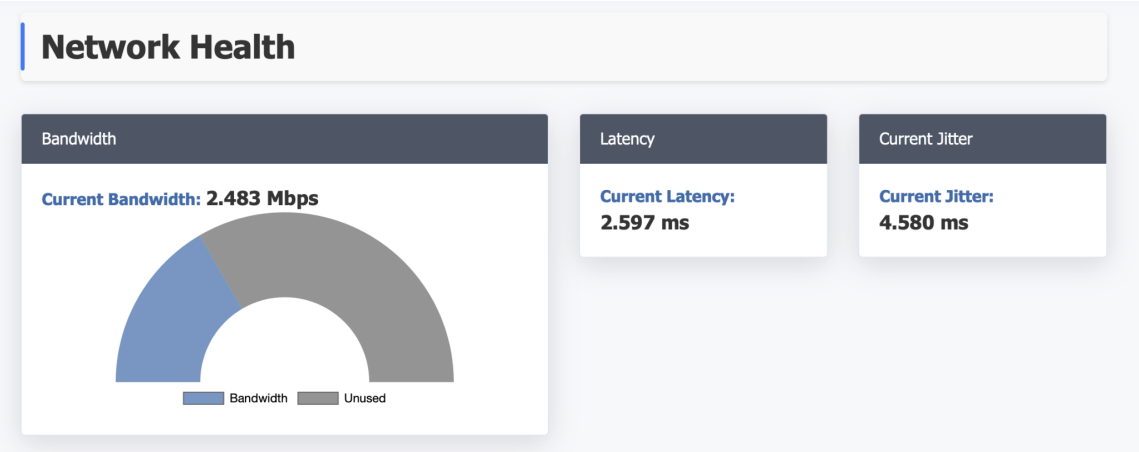

Figure 1.11: Network Health Dashboard

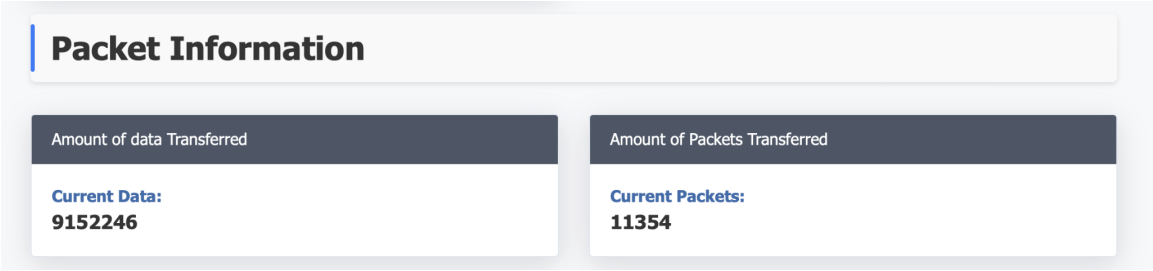

Figure 1.12: Packet Information Dashboard

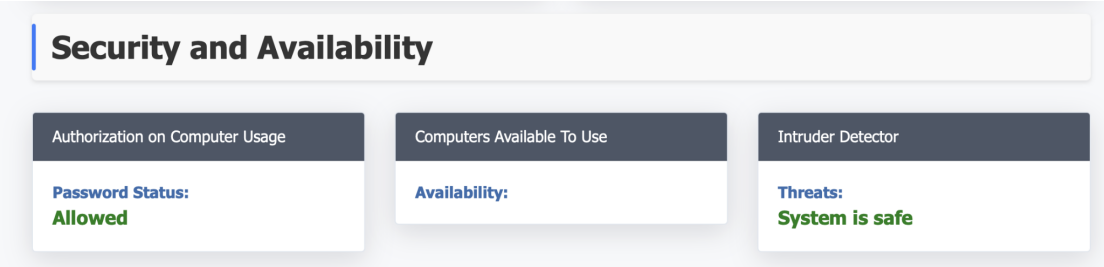

Figure 1.13: Security and Availability Dashboard

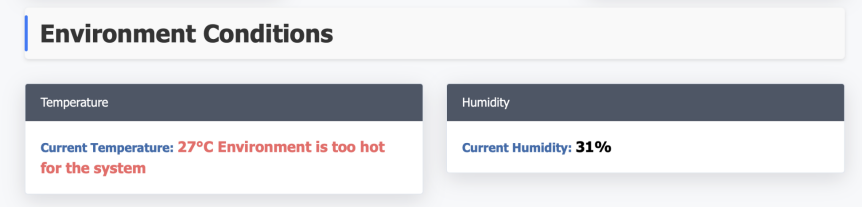

Figure 1.14: Environment Conditions Dashboard

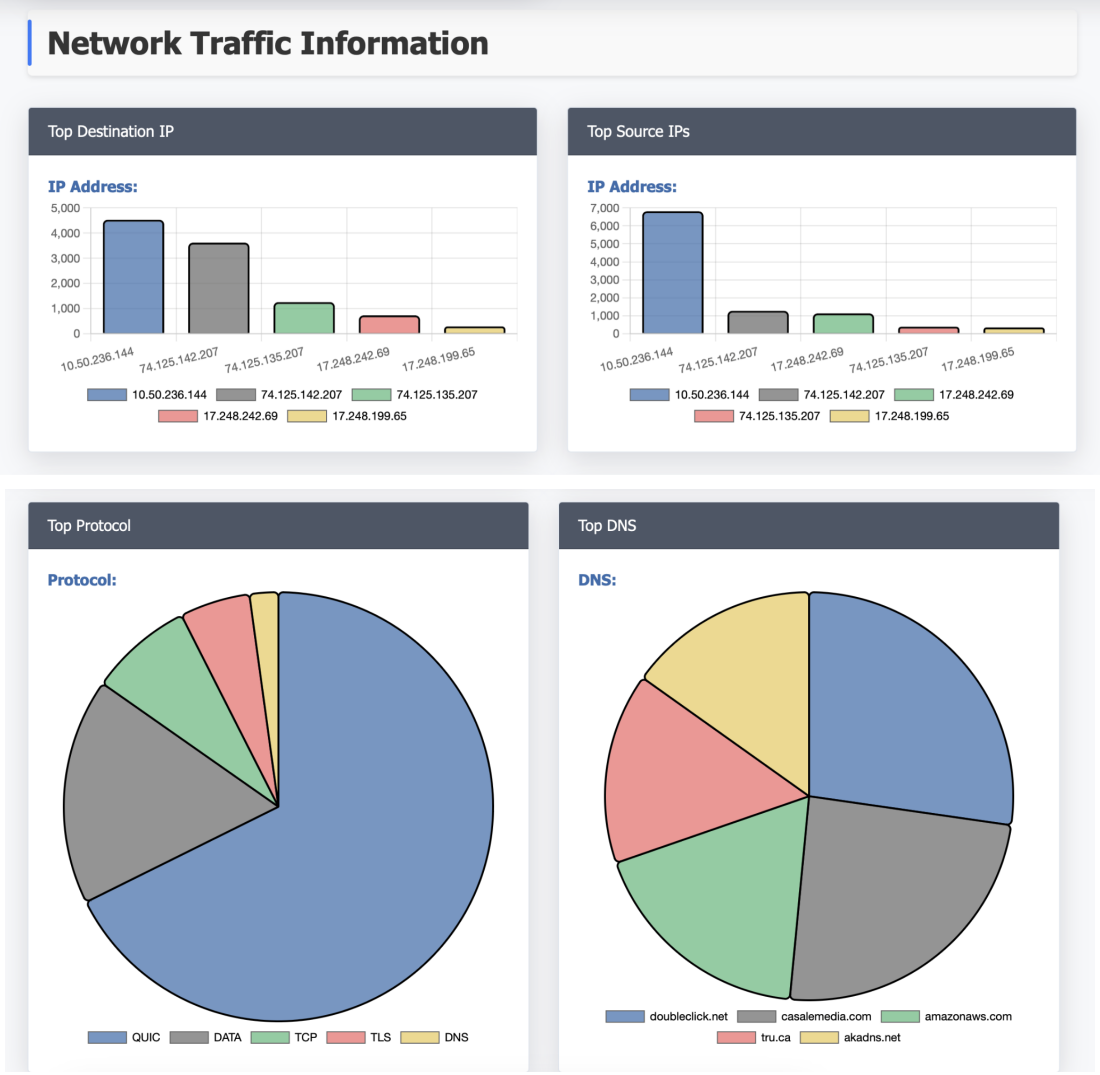

Figure 1.15: Network Traffic Information Dashboard

Results and Discussion

Results

By integrating various IoT devices, we were able to significantly broaden our system's monitoring capabilities. Developing a system with real time data collection certainly provides a better understanding of the network activities being conducted in a room. At first, Wireshark became an eye-opener as an adequate tool for packet analysis, which offered a very detailed representation of what we wanted to achieve. Similarly, the utilization of Firebase as a central database eased the storage and retrieval of our results. More so, the development of a user-friendly website proved to be a fundamental characteristic in making complex data more readable. Users could easily interpret our graphs and data visualizations, enhancing their understanding of the results.

Discussion

As a group, we believe our findings and results to be accurate and reliable since we took a careful and thorough approach to the development and implementation of this project. By testing the different aspects of our tools, we discovered the importance of creating a practical and easy-to-use system for monitoring networks and keeping them secure. Needless to say, our findings open up several gaps for improvements. Our insights certainly contribute to discussions about the effective network security solutions that exist using IoT, a growing area in both academic and practical settings. In summary, our methodology and results highlight the importance and existence of an innovative IoT based approach to develop a network monitoring and security system.

Conclusion

Conclusion

Overall, our project successfully demonstrated how an IoT-based system can be effective for measuring external conditions of a system as well as monitoring network data from the system. Various IoT devices were utilized as a response to the system’s measurements, for example, unauthorized access to the system (using the keypad) will result in a red LED turning on, or unauthorized connection to the system will indicate an intruder and sound a buzzer. Utilizing Wireshark's tools provided a comprehensive overview of all the network activities coming in and out of the system. Our user-friendly web interface made real-time graphics more accessible and increased the efficiency at which users can detect anomalies or other issues with the network. Generally speaking, the system's design promises adaptability and scalability, making it multipurpose for a variety of network environments. Our conclusions demonstrate that the system is a valuable contribution and serves as a template for systems that aim to offer a proactive approach to monitoring and securing networks.

Recommendations

Based on our findings, we suggest refining and optimizing the system itself to keep up with security threats, as it is necessary to expand on the continuous updates and improvements to the integration of said IoT devices and prevent them from being considered a liability in the network topology. Additionally, future research could focus on the integration of machine learning to further intensify the system's capabilities to adapt and respond to any network challenges encountered.

References

No external resources were used in the making of this report.

Appendix

A1: Wireshark Automated Python Program

```
import os
import pyshark
import json
from datetime import datetime
from collections import Counter
from urllib.parse import urlparse
import time
import threading
import pyrebase

# Function to extract the main domain from a URL
def extract_main_domain(url):
    parsed_url = urlparse(url)
    domain = parsed_url.netloc if parsed_url.netloc else parsed_url.path
    parts = domain.split('.')
    if len(parts) > 2:
        return '.'.join(parts[-2:])
    else:
        return domain

def upload_to_firebase():
    config ={
        "apiKey": "AlzaSyDKco5ay2xPggs8uYZ5PeBzd4rVLVEqzRM",
        "authDomain":"test-1-711d6.firebaseapp.com",
        "databaseURL":"https://test-1-711d6-default-rtdb.firebaseio.com/",
        "storageBucket":"test-1-711db6.appspot.com"
    }
    firebase = pyrebase.initialize_app(config)
    print(firebase)
    db = firebase.database()
    print(db)
    data = {}
    firebase_path_network = "NetworkTraffic"
    with open("network_traffic.json", 'r') as json_file:
        data = json.load(json_file)
```

```
    template = {
                    "NetworkTraffic": data
                    }

    db.child(firebase_path_network).set(template["NetworkTraffic"])



    print('Data uploaded successfully!')

def extract_network_traffic_info(pcap_file, output_file):
    # Open the pcap file using Pyshark
    cap = pyshark.FileCapture(pcap_file)

    # Create a list to hold the extracted data
    data = []

    # Create counters for protocols, source IP addresses, and destination
IP addresses
    protocol_counter = Counter()
    src_ip_counter = Counter()
    dest_ip_counter = Counter()
    dns_counter = Counter()

    # List of common domains to filter out
    common_domains = ['internal.example.com']

    # Initialize variables for total number of packets, total data
transfer, and timestamps
    total_packets = 0
    total_data_transfer = 0
    start_timestamp = None
    end_timestamp = None
    latencies = []
    jitter = 0
    last_latency = None
    last_timestamp = None

    # Initialize a dictionary to hold the DNS resolutions
```

```
    dns_resolutions = {}

    # Loop through each packet in the capture
    for pkt in cap:
        # Convert the timestamp to a human-readable format
        timestamp = float(pkt.sniff_timestamp)
        if start_timestamp is None or timestamp < start_timestamp:
            start_timestamp = timestamp
        if end_timestamp is None or timestamp > end_timestamp:
            end_timestamp = timestamp

        timestamp_str =
datetime.utcfromtimestamp(timestamp).strftime('%Y-%m-%d %H:%M:%S')

        # Extract the source and destination IP addresses
        src_ip = pkt.ip.src if 'IP' in pkt else 'N/A'
        dest_ip = pkt.ip.dst if 'IP' in pkt else 'N/A'

        # Extract the protocol
        protocol = pkt.highest_layer

        # Extract the length of the packet
        length = int(pkt.length)

        # Extract additional information
        info = pkt.info if 'INFO' in pkt else 'N/A'

        # If DNS packet, extract main domain
        if 'DNS' in pkt and hasattr(pkt.dns, 'qry_name'):
            dns_query_name = pkt.dns.qry_name
            main_domain = extract_main_domain(dns_query_name)
            if main_domain not in common_domains:  # Filter out common
domains
                dns_counter[main_domain] += 1
                # Store the DNS resolution in the dictionary
                if hasattr(pkt.dns, 'a'):
                    dns_resolutions[pkt.dns.a] = main_domain

        # Calculate latency and jitter
        if last_timestamp is not None:
```

```
            latency = timestamp - last_timestamp
            latencies.append(latency)
            current_jitter = abs(latency - last_latency) if last_latency
is not None else 0
            jitter += current_jitter
        else:
            latency = 0
            current_jitter = 0

        # Update last_latency and last_timestamp for next iteration
        last_latency = latency if 'latency' in locals() else None
        last_timestamp = timestamp

        # Create a dictionary with the extracted information
        packet_info = {
            'timestamp': timestamp_str,
            'src_ip': src_ip,
            'dest_ip': dest_ip,
            'protocol': protocol,
            'length': length,
            'info': info,
            'latency': f"{latency * 1e3:.3f} ms",
            'jitter': f"{current_jitter * 1e3:.3f} ms",
        }

        # Add the DNS URL to the packet info, if available
        dns_url = dns_resolutions.get(dest_ip) or
dns_resolutions.get(src_ip)
        if dns_url:
            packet_info['dns_url'] = dns_url

        # Add the dictionary to the list
        data.append(packet_info)

        # Update counters and totals
        protocol_counter[protocol] += 1
        src_ip_counter[src_ip] += 1
        dest_ip_counter[dest_ip] += 1
        total_packets += 1
        total_data_transfer += length
```

```
    # Calculate average latency and jitter
    average_latency = sum(latencies) / len(latencies) if latencies else 0
    average_jitter = jitter / (len(latencies) - 1) if len(latencies) > 1
else 0

    # Calculate bandwidth (in Mbps)
    duration = end_timestamp - start_timestamp  # in seconds
    bandwidth = (total_data_transfer * 8) / (duration * 1e6)  # in Mbps
    bandwidth = round(bandwidth, 3)  # round to 3 decimal places

    # Convert the list of dictionaries to JSON format
    json_data = json.dumps({
        'summary': {
            'total_packets': total_packets,
            'total_data_transfer': total_data_transfer,
            'bandwidth': f"{bandwidth} Mbps",
            'top_protocols': protocol_counter.most_common(5),
            'top_src_ips': src_ip_counter.most_common(5),
            'top_dest_ips': dest_ip_counter.most_common(5),
            'average_latency': f"{average_latency * 1e3:.3f} ms",
            'average_jitter': f"{average_jitter * 1e3:.3f} ms",
            'top_dns': dns_counter.most_common(30),
        }
    }, indent=4)

    # Write the JSON data to the output file
    with open(output_file, 'w') as f:
        f.write(json_data)

    # Comment indicating the extraction has finished
    print(f"Finished extracting data to {output_file}")

    cap.close()  # Close the capture file to free resources

# Function to capture network traffic
def capture_packets(interface='en0', duration=30,
output_file='network_traffic.pcap'):
    print("===== CAPTURING PACKETS =====")
```

```
    capture = pyshark.LiveCapture(interface=interface,
output_file=output_file)

    try:
        capture.sniff(timeout=duration)
        print("===== PACKET CAPTURE COMPLETED =====")
    except Exception as e:
        print(f"An error occurred during packet capture: {e}")
    finally:
        capture.close()
        print(f"Packets saved to {output_file}")
        # After capture, process the packets immediately
        json_output_file = output_file.replace('.pcap', '.json')
        extract_network_traffic_info(output_file, json_output_file)

# Function to stop capturing based on user input
def wait_for_enter():
    input("Press Enter to stop capturing at the next opportunity...\n")
    global capture_flag
    capture_flag = False




# The main loop to start capturing and processing
def main(interface='wlan0', capture_duration=30):
    global capture_flag
    capture_flag = True
    thread = threading.Thread(target=wait_for_enter)
    thread.start()

    # Define the filenames for pcap and JSON, which will be overwritten
each time
    pcap_filename = 'network_traffic.pcap'
    json_filename = 'network_traffic.json'

    while capture_flag:

        capture_packets(interface, capture_duration, pcap_filename)
        upload_to_firebase()
```

```
        # Sleep or perform other tasks here if necessary
        time.sleep(1)  # Short sleep to allow for thread context switching

    thread.join()
    print("===== PROGRAM STOPPED BY USER =====")

if __name__ == "__main__":
    main()
```

A2: Keypad Authorization Python Program

```
import RPi.GPIO as GPIO
import time
import pyrebase
from gpiozero import LED
import subprocess

def run_external_program(program_path):
    try:
        subprocess.Popen(['lxterminal', '-e', 'python', program_path])
    except Exception as e:
        print(f"An error occurred while running {program_path}: {e}")

config ={
    "apiKey": "AlzaSyDKco5ay2xPggs8uYZ5PeBzd4rVLVEqzRM",
    "authDomain":"test-1-711d6.firebaseapp.com",
    "databaseURL":"https://test-1-711d6-default-rtdb.firebaseio.com",
    "storageBucket":"test-1-711db6.appspot.com"
}

firebase = pyrebase.initialize_app(config)
db = firebase.database()

led_granted = LED(24)
led_denied = LED(23)

firebase_path_password = "Password"

first_input_received = False  # Flag for the first input

def stream_handler(message):
    global first_input_received
    if not first_input_received:
        print("Welcome to the admin program manager. Please enter the
correct code to run the programs.")

        return

    print("Checking the entered code...")
```

```
    if message["data"] == "True":
        print("Access Granted. Activating granted LED...")
        led_granted.on()
        time.sleep(3)
        led_granted.off()
        print("Running external programs...")
        run_external_program('automatedNetworkTraffic.py')
        run_external_program('buzzer.py')
        run_external_program('proximity.py')
        run_external_program('temperature.py')
    else:
        print("Access Denied. Activating denied LED...")
        print("Please try again...")
        led_denied.on()
        led_granted.off()
        time.sleep(3)
        led_denied.off()

    first_input_received = False  # Reset the flag after checking the
password

my_stream = db.child("Allowed").stream(stream_handler)

L1, L2, L3, L4 = 5, 6, 13, 19
C1, C2, C3, C4 = 12, 16, 20, 21

keypadPressed = -1
input = ""

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup([L1, L2, L3, L4], GPIO.OUT)
GPIO.setup([C1, C2, C3, C4], GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

def keypadCallback(channel):
    global keypadPressed
    if keypadPressed == -1:
        keypadPressed = channel
```

```
GPIO.add_event_detect(C1, GPIO.RISING, callback=keypadCallback)
GPIO.add_event_detect(C2, GPIO.RISING, callback=keypadCallback)
GPIO.add_event_detect(C3, GPIO.RISING, callback=keypadCallback)
GPIO.add_event_detect(C4, GPIO.RISING, callback=keypadCallback)

def setAllLines(state):
    GPIO.output([L1, L2, L3, L4], state)

def checkSpecialKeys():
    global input, first_input_received
    pressed = False

    GPIO.output(L3, GPIO.HIGH)
    if GPIO.input(C4) == 1:
        print("Input Reset!")
        input = ""
        pressed = True
    GPIO.output(L3, GPIO.LOW)

    GPIO.output(L1, GPIO.HIGH)
    if not pressed and GPIO.input(C4) == 1:
        db.child(firebase_path_password).set(input)
        pressed = True
        print("Password Sent")
    GPIO.output(L1, GPIO.LOW)

    if pressed:
        input = ""
        first_input_received = False  # Reset the flag

    return pressed

def readLine(line, characters):
    global input
    GPIO.output(line, GPIO.HIGH)
    for i, char in enumerate(characters):
        if GPIO.input([C1, C2, C3, C4][i]) == 1:
            print(char)
            input += char
    GPIO.output(line, GPIO.LOW)
```

```
try:
    while True:
        if keypadPressed != -1:
            first_input_received = True  # Set the flag when first input
is detected
            setAllLines(GPIO.HIGH)
            if GPIO.input(keypadPressed) == 0:
                keypadPressed = -1
            time.sleep(0.1)
        else:
            if not checkSpecialKeys():
                readLine(L1, ["1", "2", "3", "A"])
                readLine(L2, ["4", "5", "6", "B"])
                readLine(L3, ["7", "8", "9", "C"])
                readLine(L4, ["*", "0", "#", "D"])
            time.sleep(0.1)
except KeyboardInterrupt:
    print("\nApplication Stopped.")
    GPIO.cleanup()  # Clean up GPIO on CTRL+C exit
```

A3: Intruder Buzzer Activation Python Program

```
from gpiozero import Buzzer
import time
import RPi.GPIO as GPIO
import pyrebase

config ={
     "apiKey": "AlzaSyDKco5ay2xPggs8uYZ5PeBzd4rVLVEqzRM",
     "authDomain":"test-1-711d6.firebaseapp.com",
     "databaseURL":"https://test-1-711d6-default-rtdb.firebaseio.com",
     "storageBucket":"test-1-711db6.appspot.com"
}
firebase = pyrebase.initialize_app(config)
print(firebase)
db = firebase.database()
print(db)

buzzer = Buzzer(27)


def stream_handler(message):
     print("Checking for intruder")
     if(message["data"] == "True"):
          buzzer.on()
          print("INTRUDER DETECTED")
     else:
          buzzer.off()
          print("INTRUDER NEUTRALIZE, SYSTEM SAFE")


my_stream = db.child("Intruder").stream(stream_handler)
```

A4: User Proximity Python Program

```
from gpiozero import Buzzer
import time
import RPi.GPIO as GPIO
import pyrebase

config ={
    "apiKey": "AlzaSyDKco5ay2xPggs8uYZ5PeBzd4rVLVEqzRM",
    "authDomain":"test-1-711d6.firebaseapp.com",
    "databaseURL":"https://test-1-711d6-default-rtdb.firebaseio.com",
    "storageBucket":"test-1-711db6.appspot.com"
}
firebase = pyrebase.initialize_app(config)
print(firebase)
db = firebase.database()
print(db)

firebase_availability_path = "Availability"
flag = False


GPIO_TRIGGER = 17
GPIO_ECHO = 4

buzzer = Buzzer(27)

GPIO.setup(GPIO_TRIGGER, GPIO.OUT)
GPIO.setup(GPIO_ECHO, GPIO.IN)

def distance():
    #Set Trigger to HIGH
    GPIO.output(GPIO_TRIGGER, True)

    #Set Trigger after 0.01ms to LOW
    time.sleep(0.00001)
    GPIO.output(GPIO_TRIGGER, False)

    StartTime = time.time()
    StopTime = time.time()
```

```
    #Save StartTime
    while GPIO.input(GPIO_ECHO) == 0:
        StartTime = time.time()

    #Save time of arrival
    while GPIO.input(GPIO_ECHO) == 1:
        StopTime = time.time()

    #Time difference between start and arrival
    TimeElapsed = StopTime - StartTime

    #Multiply with the sonic speed (34300 cm/s) and divide by 2, because
there and back
    distance = (TimeElapsed * 34300) / 2
    return distance

if __name__ == '__main__':
    data = 1
    while True:

        dist = distance()
        print("Measured Distance = %.1f cm" % dist)
        time.sleep(1)


        if(dist < 100):

            template = {
                "Availability": str(data)

            }
            if(not flag):
                data = 0
                db.child(firebase_availability_path).set(str(data))
                flag = True
                print("data sent")
```

```
        else:
             if(flag):
                flag = False
                x = {"Availability" : data}
                data = 1
                db.child(firebase_availability_path).set(str(data))
```

A5: Environmental Conditions Python Program

```
import RPi.GPIO as GPIO
import Adafruit_DHT
import time
import pyrebase


DHT_SENSOR = Adafruit_DHT.DHT11
DHT_PIN = 25

config ={
     "apiKey": "AlzaSyDKco5ay2xPggs8uYZ5PeBzd4rVLVEqzRM",
     "authDomain":"test-1-711d6.firebaseapp.com",
     "databaseURL":"https://test-1-711d6-default-rtdb.firebaseio.com",
     "storageBucket":"test-1-711db6.appspot.com"
}
firebase = pyrebase.initialize_app(config)
print(firebase)
db = firebase.database()
print(db)
firebase_path_temp = "temperature"
firebase_path_humid = "humidity"

temp_data=0.0
humid_data=0.0

try:
     while True:
          # Read the temperature and humidity from the DHT sensor
          humidity,temperature = Adafruit_DHT.read(DHT_SENSOR,DHT_PIN)

          if humidity is not None and temperature is not None:
               print(f"Temperature: {temperature:.1f}C,Humidity :
{humidity:.1f}%")

               template_temp = {
                    "temperature": temperature

                    }
```

```
                template_humid = {
                     "humidity": humidity

                     }

                db.child(firebase_path_temp).set(temperature)
                db.child(firebase_path_humid).set(humidity)


          time.sleep(2)#delay for 5 seconds between readings


except KeyboardInterrupt:
     pass

finally:
     #clean up GPI and PWM when the script is topped
     pwm.stop()
     GPIO.cleanup()
```